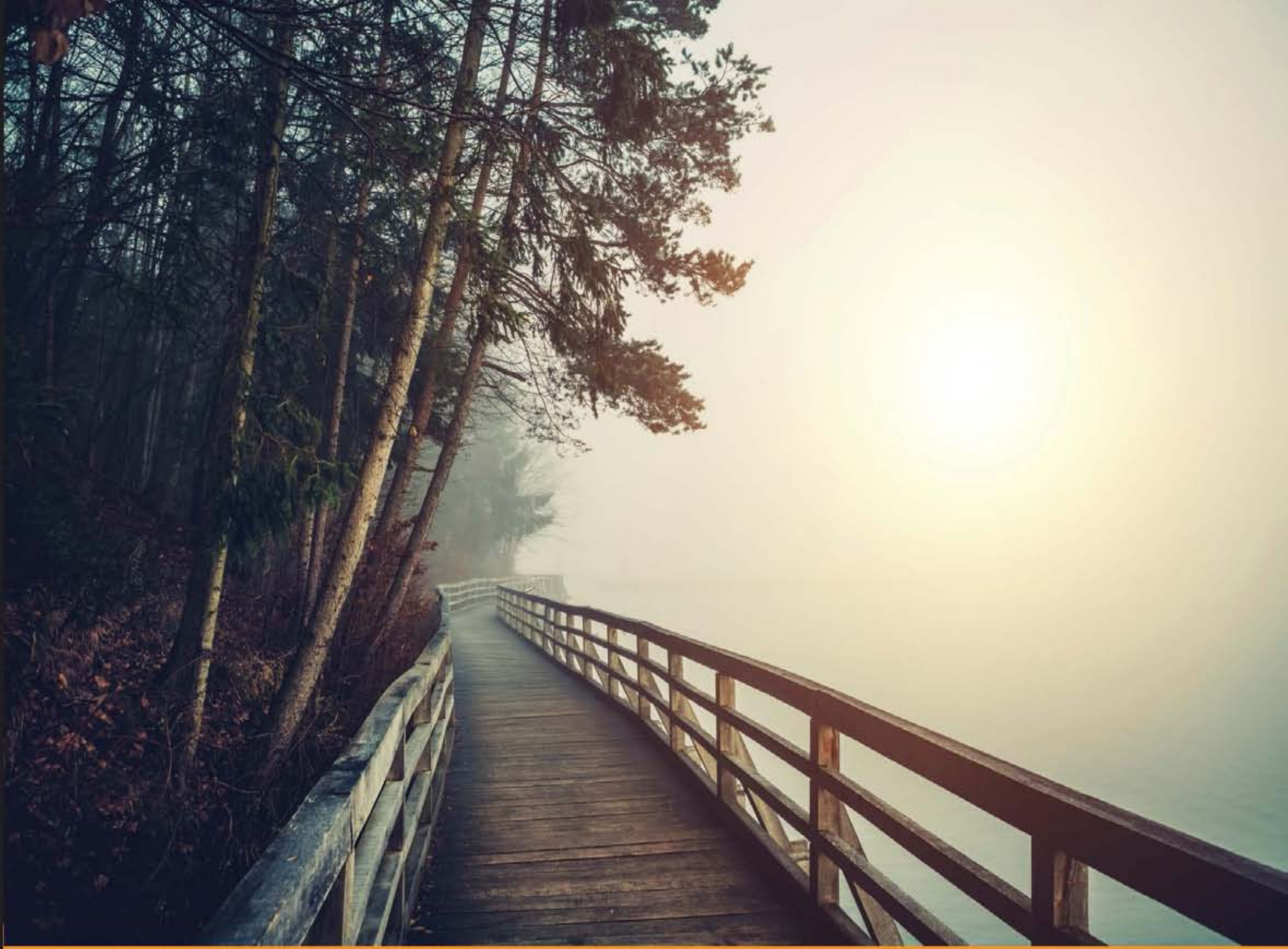

Community Experience Distilled

Mastering Yii

Advance your modern web application development skills with Yii Framework 2

Charles R. Portwood II

[PACKT] open source*
PUBLISHING
community experience distilled

Mastering Yii

Advance your modern web application development skills with Yii Framework 2

Charles R. Portwood II

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Mastering Yii

First published: January 2016

Production reference: 1210116

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78588-242-5

www.packtpub.com

Credits

Author
Charles R. Portwood II

Reviewer
Tomasz Trejderowski

Acquisition Editor
Divya Poojari

Content Development Editor
Anish Dhurat

Technical Editor
Edwin Moses

Copy Editor
Stuti Srivastava

Project Coordinator
Bijal Patel

Proofreader
Safis Editing

Indexer
Priya Sane

Production Coordinator
Shantanu N. Zagade

Cover Work
Shantanu N. Zagade

About the Author

Charles R. Portwood II has over 10 years of experience developing modern web applications and is well versed in integrating PHP with native mobile applications. An avid proponent of Yii Framework and open source software, Charles has contributed multiple guides, extensions, and applications to the Yii community. In addition to being a programmer, he is also a Linux system administrator.

About the Reviewer

Tomasz Trejderowski is a middle-aged developer from Poland who has hands-on experience working with many programming languages and in diverse IT-related areas. He has been programming computers since the very first Commodore 64 and thus, he poses over 20 years of software development experience. You can access repositories and contributions on his GitHub profile, at `http://github.com/trejder`.

He is a full-time business analyst and free-time PhoneGap/Yii2 developer and blogger. He is also a mobile market entrepreneur, constantly working on some innovative projects. For more information, visit his company website at `http://www.gaman.pl` or his blog network at `http://www.acrid.pl/`.

He is a happy husband of his wonderful wife and father of two beautiful daughters.

Table of Contents

Preface

Yii Framework 2 (Yii2) is the successor to the popular Yii framework. Like its successor, Yii2 is an open source, high-performance rapid development framework designed to create modern, scalable, and performant web applications and APIs.

Designed for both developers with no exposure to Yii and Yii2 and for Yii framework developers looking to become experts with Yii2, this book will serve as your guide to becoming a master of Yii. From initialization and configuration to debugging and deployment, this book will be your guide to becoming a master of all aspects of this powerful framework.

What this book covers

Chapter 1, *Composer, Configuration, Classes, and Path Aliases*, covers the basics of a Yii2 application. In this chapter, you'll learn the core conventions of Yii2 and how to configure it as a multi-environment application. You'll also discover how to use Composer, a dependency management tool for managing your applications' software dependencies.

Chapter 2, *Console Commands and Applications*, focuses on how to use the built-in Yii2 console commands as it guides you through creating your own commands.

Chapter 3, *Migrations, DAO, and Query Building*, teaches you how to create migrations in Yii2 and how to interact with your database using database access objects (DAO) and how to use Yii2's query builder.

Chapter 4, *Active Record, Models, and Forms*, teaches you how to create and use Active Record to effortlessly interact with a database. Furthermore, you'll also discover how to create models to represent information not stored in databases and how to create web forms based upon Active Record models and normal models.

Chapter 5, *Modules, Widgets, and Helpers*, covers how to incorporate modules inside of our application. This chapter will also cover how to create and use dynamic widgets and will additionally cover Yii2's powerful helper classes.

Chapter 6, *Asset Management*, focuses on how to create and manage our assets using asset bundles and how to manage our assets using the asset command. This chapter also covers several strategies to build and generate our asset library using powerful tools such as Node Package Manage and Bower.

Chapter 7, *Authenticating and Authorizing Users*, teaches you how to verify the authenticity of users in Yii2 using several common authentication schemes (such as OAuth authentication, basic HTTP authentication, and header authentication) as well as shows you how to grant them access to specific sections of your applications.

Chapter 8, *Routing, Responses, and Events*, focuses on how Yii2's routing and response classes work in Yii2. In this chapter, we'll cover how to handle data both in and out of our application and discover how to tap into Yii2's powerful event system.

Chapter 9, *RESTful APIs*, talks about how to quickly and effortlessly extend your application with a RESTful JSON and XML API using Yii2's ActiveController class.

Chapter 10, *Testing with Codeception*, helps you learn how to create unit, functional, and acceptance tests for your applications using a powerful testing tool called Codeception. In this chapter, you'll also learn how to create fixtures to represent your data for testing purposes.

Chapter 11, *Internationalization and Localization*, covers how to localize our applications and build them to support multiple languages. Additionally, you will master how to create and manage translation files using Yii2 console commands.

Chapter 12, *Performance and Security*, covers many ways to improve the performance of your Yii2 application and how to keep it secure against modern day attacks on web applications.

Chapter 13, *Debugging and Deploying*, helps you become well-versed in how to debug your Yii2 applications using both application logging and the Yii2 debug tool. Furthermore, you will discover the fundamentals of deploying your Yii2 applications in a seamless and non-disruptive fashion.

What you need for this book

To ensure a consistent development environment and prevent unnecessary alterations to your host operation system, it is highly recommended that you run all commands within a Linux virtual machine. This will ensure that your output both in your web browser and from your command line matches the output that is presented in this book. As setting up this environment on your own can be a daunting task, prebuilt virtual machines that use VirtualBox and Vagrant are provided to make this setup process easy.

To get started with this book, you should be running the latest version of either Microsoft Windows 7, 8, 8.1 or 10, Apple OS X 10.9 or higher, or a Linux operating system that can run virtual machines, such as Ubuntu 14.04 LTS. Additionally, you will need to install the latest version of VirtualBox (available at `https://www.virtualbox.org/wiki/Downloads`) and Vagrant (available at `https://www.vagrantup.com/downloads.html`).

[After installing these software dependencies, you may need to restart your computer for the changes to take effect.]

After installing VirtualBox and Vagrant, you can then create a new dedicated development environment by opening a new command line or terminal window, creating a new directory for the chapter, and then running the following command to create your virtual machine development environment. These commands will download a prebuilt virtual machine containing all the software required to get you started and start your new development environment:

```
vagrant init charlesportwoodii/php56_trusty64
vagrant up --provider virtualbox
vagrant ssh
```

[More information on this specific Vagrant box can be found at `https://atlas.hashicorp.com/charlesportwoodii/boxes/php56_trusty64`.

Note that if you are on Windows, you may need a tool such as PuTTy to connect to your virtual machine over SSH. More information on how to connect to your new virtual machine over SSH on Windows can be found at `http://docs-v1.vagrantup.com/v1/docs/getting-started/ssh.html`.]

Once your new Vagrant box has started, you can access the files of this virtual machine over SSH and access your `webroot` directory by opening a new browser window and navigating to `http://localhost:8080`. By default, when you open this web page, you will see the output of `phpino()`.

> Depending upon your operating system security settings, your computer may prompt or block you from accessing port 8080 on your computer. Ensure that you configure your firewall settings if you are facing issues and ensure that port 8080 is open on your computer and that VirtualBox can forward connections from your host operating system to your guest operating system.

As Yii2 is fully compatible with PHP7, it is strongly suggested that you develop and test your web applications against PHP7 as well. The following commands will allow you to provision a PHP7 Vagrant box:

```
vagrant init charlesportwoodii/php7_trusty64
vagrant up --provider virtualbox
vagrant ssh
```

> As these virtual machines automatically configure port forwarding, it is recommended that you only run a single virtual machine at a time. Refer to the Vagrant documentation for a complete list of commands and configuration options at `https://docs.vagrantup.com/v2`.

Who this book is for

Mastering Yii is for intermediate to experienced software developers who want to quickly master Yii2. This book assumes some familiarity with PHP 5, HTML5, and rudimentary software development practices and methodologies.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: " This script tells Composer that when the `create-project` command is run, it should run the `postCreateProject` static function."

A block of code is set as follows:

```
"scripts": {
    "post-create-project-cmd": [
        "yii\\composer\\Installer::postCreateProject"
    ]
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
// Define our application_env variable as provided by nginx/apache
if (!defined('APPLICATION_ENV'))
{
    if (getenv('APPLICATION_ENV') != false)
        define('APPLICATION_ENV', getenv('APPLICATION_ENV'));
    else
       define('APPLICATION_ENV', 'prod');
}

$env = require(__DIR__ . '/config/env.php');
```

Any command-line input or output is written as follows:

```
$ ./yii fixture/load <FixtureName>
$ ./yii fixture/unload <FixtureName>
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: " Once we have specified all the necessary attributes, we can click on the **Preview** button to preview our form, and then we can click on the **Generate** button to generate the source code."

[Warnings or important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

The latest and most up to date copies of source code for this book is maintained on the Packt website: `http://www.packtpub.com` and on GitHub at `https://github.com/masteringyii`, for each chapter where applicable.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Composer, Configuration, Classes, and Path Aliases

Before diving into Yii Framework 2, we need to take a look at how it is installed, how it is configured, and what the core building blocks of the framework are. In this chapter, we'll go over how to install the framework itself and prebuilt applications via a package management tool called **Composer**. We'll also cover some common configurations of both Yii Framework 2 and our web server, including making our applications aware of the environment they are running on and responding appropriately to that environment.

[The most common ways to reference Yii Framework 2 are *Yii Framework 2*, *YF2*, and *Yii2*. We'll be using these terms interchangeably throughout the book.]

Composer

There are several different ways to install Yii2, ranging from downloading the framework from source control (typically, from GitHub at `https://github.com/yiisoft/yii2`) to using a package manager such as Composer. With modern web applications, Composer is the preferred method to install Yii2 as it enables us to install, update, and manage all dependencies and extensions for our application in an automated fashion. Additionally, using Composer, we can ensure that Yii Framework 2 is kept up to date with the latest security and bug fixes. Composer can be installed by following the instructions on `https://getcomposer.org`. Typically, this process looks as follows:

```
curl -sS https://getcomposer.org/installer | php
```

Alternatively, if you don't have cURL available on your system, it can be installed through PHP itself:

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

Once installed, we should move Composer to a more centralized directory so that we can call it from any directory on our system. Installing Composer from a centralized directory rather than on a per-project basis has several advantages:

- It can be called anywhere from any project. When working with multiple projects, we can ensure that we use the same dependency manager each time and for every project.
- In a centralized directory, Composer only needs to be updated once rather than in every project we are working on.
- Dependency managers are rarely considered code that should be pushed to your DCVS repository. Keeping the `composer.phar` file out of your repository reduces the amount of code you need to commit and push and ensures that your source code remains isolated from your package manager code.
- By installing Composer from a centralized directory, we can ensure that Composer is always available, which saves us a step each time we clone a project that depends on Composer.

A good directory to move Composer to is `/usr/local/bin`, as shown in the following example:

```
mv composer.phar /usr/local/bin/composer
chmod a+x /usr/local/bin/composer
```

Throughout this book, we'll be using Unix-style commands when referencing command-line arguments. Consequently, some commands may not work on Windows. If you decide to set up a Windows environment, you might need to use `Composer-Setup.exe` (available at `https://getcomposer.org/Composer-Setup.exe`) to get Composer configured for your system. If you have any issues getting Composer to run on your system, ensure that you check out the Composer documentation available at `https://getcomposer.org/doc/`.

Alternatively, if you have Composer installed on your system already, ensure that you update it to the latest version by running this:

```
composer self-update
```

The commands that we use through this book are based on the assumption that you have sufficient privileges to run them. On Unix-like systems, you may need to preface some commands with `sudo` in order to execute the command with a high permissions set. Alternatively if you are running these commands on Windows, you should ensure that you are running the listed commands in a command prompt that has elevated privileges. Ensure that you follow best practices when using `sudo` and when using elevated command prompts in order to ensure your system stays secure.

Once Composer is installed, we'll need to install a global plugin called **The Composer Asset Plugin** (available at `https://github.com/francoispluchino/composer-asset-plugin`). This plugin enables Composer to manage asset files for us without the need to install additional software (these programs are Bower, an asset dependency manager created by Twitter, and Node Package Manager, or NPM, which is a JavaScript dependency manager).

```
composer global require "fxp/composer-asset-plugin:1.0.0"
```

Due to the GitHub API's rate limiting, during installation, Composer may ask you to enter your GitHub credentials. After entering your credentials, Composer will request a dedicated API key from GitHub that can be used for future installations. Ensure that you check out the Composer documentation at `https://getcomposer.org/doc/` for more information.

With Composer installed, we can now instantiate our application. If we want to install an existing Yii2 package, we can simply run the following:

```
composer create-project --prefer-dist <package/name> <foldername>
```

Using the Yii2 basic app as an example, this command will look like this:

```
composer create-project --prefer-dist yiisoft/yii2-app-basic basic
```

After running the command, you should see output similar to the following:

```
Installing yiisoft/yii2-app-basic (2.0.6)
  - Installing yiisoft/yii2-app-basic (2.0.6)
    Downloading: 100%
Created project in basic
Loading composer repositories with package information
Installing dependencies (including require-dev)
```

```
  - Installing yiisoft/yii2-composer (2.0.3)
  - Installing ezyang/htmlpurifier (v4.6.0)
  - Installing bower-asset/jquery (2.1.4)
  - Installing bower-asset/yii2-pjax (v2.0.4)
  - Installing bower-asset/punycode (v1.3.2)
  - Installing bower-asset/jquery.inputmask (3.1.63)
  - Installing cebe/markdown (1.1.0)
  - Installing yiisoft/yii2 (2.0.6)
  - Installing swiftmailer/swiftmailer (v5.4.1)
  - Installing yiisoft/yii2-swiftmailer (2.0.4)
  - Installing yiisoft/yii2-codeception (2.0.4)
  - Installing bower-asset/bootstrap (v3.3.5)
  - Installing yiisoft/yii2-bootstrap (2.0.5)
  - Installing yiisoft/yii2-debug (2.0.5)
  - Installing bower-asset/typeahead.js (v0.10.5)
  - Installing phpspec/php-diff (v1.0.2)
  - Installing yiisoft/yii2-gii (2.0.4)
  - Installing fzaninotto/faker (v1.5.0)
  - Installing yiisoft/yii2-faker (2.0.3)
Writing lock file
Generating autoload files
> yii\composer\Installer::postCreateProject
chmod('runtime', 0777)...done.
chmod('web/assets', 0777)...done.
chmod('yii', 0755)...done.
```

Your output may differ slightly due to the data cached on your system and versions of subpackages.

This command will install the Yii2 basic app to a folder called basic. When creating a new Yii2 project, you'll typically want to use the create-project command to clone "yii2-app-basic" and then develop your application from there as the basic app comes prepopulated with just about everything you need to start a new project. However, you can also create a Yii2 project from scratch that, while more complicated, gives you more control over your application's structure.

Let's take a look at the `composer.json` file that was created when we ran the `create-project` command:

```
{
    "name": "yiisoft/yii2-app-basic",
    "description": "Yii 2 Basic Application Template",
    "keywords": ["yii2", "framework", "basic",
    "application template"],
    "homepage": "http://www.yiiframework.com/",
    "type": "project",
    "license": "BSD-3-Clause",
    "support": {
        "issues": "https://github.com/
        yiisoft/yii2/issues?state=open",
        "forum": "http://www.yiiframework.com/forum/",
        "wiki": "http://www.yiiframework.com/wiki/",
        "irc": "irc://irc.freenode.net/yii",
        "source": "https://github.com/yiisoft/yii2"
    },
    "minimum-stability": "stable",
    "require": {
        "php": ">=5.4.0",
        "yiisoft/yii2": "*",
        "yiisoft/yii2-bootstrap": "*",
        "yiisoft/yii2-swiftmailer": "*"
    },
    "require-dev": {
        "yiisoft/yii2-codeception": "*",
        "yiisoft/yii2-debug": "*",
        "yiisoft/yii2-gii": "*",
        "yiisoft/yii2-faker": "*"
    },
    "config": {
        "process-timeout": 1800
    },
    "scripts": {
        "post-create-project-cmd": [
            "yii\\composer\\Installer::postCreateProject"
        ]
    },
    "extra": {
        "yii\\composer\\Installer::postCreateProject": {
            "setPermission": [
                {
                    "runtime": "0777",
                    "web/assets": "0777",
                    "yii": "0755"
                }
            ],
            "generateCookieValidationKey": [
                "config/web.php"
            ]
        },
        "asset-installer-paths": {
            "npm-asset-library": "vendor/npm",
            "bower-asset-library": "vendor/bower"
        }
    }
}
```

While most of these items (such as the name, description, license, and require blocks) are rather self-explanatory, there are a few Yii2-specific items in here that we should take note of. The first section we want to look at is the `"scripts"` section:

```
"scripts": {
    "post-create-project-cmd": [
        "yii\\composer\\Installer::postCreateProject"
    ]
}
```

This script tells Composer that when the `create-project` command is run, it should run the `postCreateProject` static function. Looking at the the framework source code, we see that this file is referenced in the `yii2-composer` package (refer to `https://github.com/yiisoft/yii2-composer/blob/master/Installer.php#L232`). This command then runs several post-project creation actions, namely setting the local disk permissions, generating a unique cookie validation key, and setting some asset installer paths for composer-asset-plugin.

Next, we have the `"extra"` block:

```
"extra": {
    "yii\\composer\\Installer::postCreateProject": {
        "setPermission": [
            {
                "runtime": "0777",
                "web/assets": "0777",
                "yii": "0755"
            }
        ],
        "generateCookieValidationKey": [
            "config/web.php"
        ]
    },
    "asset-installer-paths": {
        "npm-asset-library": "vendor/npm",
        "bower-asset-library": "vendor/bower"
    }
}
```

This section tells Composer to use these options when it runs the `postCreateProject` command. These preconfigured options give us a good starting point to create our applications.

Configuration

With our basic application now installed, let's take a look at a few basic configuration and bootstrap files that Yii2 automatically generated for us.

Requirements checker

Projects created from `yii2-app-basic` now come with a built-in requirements script called `requirements.php`. This script checks several different values in order to ensure that Yii2 can run on our application server. Before running our application, let's run the requirements checker:

```
php requirements.php
```

You'll get output similar to the following:

```
Yii Application Requirement Checker

This script checks if your server configuration meets the requirements
for running Yii application.

It checks if the server is running the right version of PHP,  if
appropriate PHP extensions have been loaded, and if php.ini file settings
are correct.

Check conclusion:

-----------------

PHP version: OK

[... more checks here ...]

-----------------------------------------

Errors: 0   Warnings: 6   Total checks: 21
```

In general, as long as the error count is set to `0`, we'll be good to move forward. If the requirements checker notices an error, it will report it in the `Check conclusion` section for you to rectify.

> As part of your deployment process, it is recommended that your deployment tool runs the requirements checker. This helps ensure that your application server meets all the requirements for Yii2 and that your application doesn't get deployed to a server or environment that doesn't support it.

Entry scripts

Like its predecessor, Yii Framework 2 comes with two separate entry scripts: one for web applications and the other for console applications.

Web entry script

In Yii2, the entry script for web applications has been moved from the root (/) folder to the `web/` folder. In Yii1, our PHP files were stored in the `protected/` directory. By moving our entry scripts to the `web/` directory, Yii2 has increased the security of our application by reducing the amount of web server configuration we need to run our application. Furthermore, all public asset (JavaScript and CSS) files are now completely isolated from our source code directories. If we open up `web/index.php`, our entry script now looks as follows:

```
<?php

// comment out the following two lines when deployed to production
defined('YII_DEBUG') or define('YII_DEBUG', true);
defined('YII_ENV') or define('YII_ENV', 'dev');

require(__DIR__ . '/../vendor/autoload.php');
require(__DIR__ . '/../vendor/yiisoft/yii2/Yii.php');

$config = require(__DIR__ . '/../config/web.php');

(new yii\web\Application($config))->run();
```

> **Downloading the example code**
> The latest and most up to date copies of source code for this book is maintained on the Packt Publishing website, `http://www.packtpub.com`, and on `GitHub at https://github.com/masteringyii`, for each chapter where applicable.

While suitable for basic applications, the default entry script requires us to manually comment out and change the code when moving to different environments. Since changing the code in a nondevelopment environment doesn't follow best practices, we should change this code block so that we don't have to touch our code to move it to a different environment.

We'll start by creating a new application-wide constant called `APPLICATION_ENV`. This variable will be defined by either our web server or our console environment and will allow us to dynamically load different configuration files depending upon the environment that we're working in:

1. After the opening `<?php` tag in `web/index.php`, add the following code block:

   ```
   // Define our application_env variable as provided by nginx/
   apache/console
   if (!defined('APPLICATION_ENV'))
   {
       if (getenv('APPLICATION_ENV') != false)
           define('APPLICATION_ENV',
           getenv('APPLICATION_ENV'));
       else
          define('APPLICATION_ENV', 'prod');
   }
   ```

   Our application now knows how to read the `APPLCATTION_ENV` variable from the environment variable, which will be passed either though our command line or our web server configuration. By default, if no environment is set, the `APPLICATION_ENV` variable will be set to prod.

   Next, we'll want to load a separate environment file that contains several environmental constants that we'll use to dynamically change how our application runs in different environments:

   ```
   $env = require(__DIR__ . '/../config/env.php');
   ```

   Next, we'll configure Yii to set the `YII_DEBUG` and `YII_ENV` variables according to our application:

   ```
   defined('YII_DEBUG') or define('YII_DEBUG', $env['debug']);
   defined('YII_ENV') or define('YII_ENV', APPLICATION_ENV);
   ```

2. Then, follow the rest of our `index.php` file under `web/`:

   ```
   require(__DIR__ . '/../vendor/autoload.php');
   require(__DIR__ . '/../vendor/yiisoft/yii2/Yii.php');
   (new yii\web\Application($config))->run();
   ```

With these changes, our web application is now configured to be aware of its environment and load the appropriate configuration files.

Don't worry; later in the chapter, we'll cover how to define the `APPLICATION_ENV` variable for both our web server (either Apache or NGINX) and our command line.

Configuration files

In Yii2, configuration files are still split into console- and web-specific configurations. As there are many commonalities between these two files (such as our database and environment configuration), we'll store common elements in their own files and include those files in both our web and console configurations. This will help us follow the DRY standard, and reduce duplicate code within our application.

The **DRY** (**don't repeat yourself**) principle in software development states that we should avoid having the same code block appear in multiple places in our application. By keeping our application DRY, we can ensure that our application is performant and can reduce bugs in our application. By moving our database and parameters' configuration to their own file, we can reuse that same code in both our web and console configuration files.

Web and console configuration files

Yii2 supports two different kinds of configuration files: one for web applications and another for console applications. In Yii2, our web configuration file is stored in `config/web.php` and our console configuration file is stored in `config/console.php`. If you're familiar with Yii1, you'll see that the basic structure of both of these files hasn't changed all that much.

Database configuration

The next file we'll want to look at is our database configuration file stored in `config/db.php`. This file contains all the information our web and console applications will need in order to connect to the database.

In our basic application, this file looks as follows:

```
<?php

return [
    'class' => 'yii\db\Connection',
    'dsn' => 'mysql:host=localhost;dbname=yii2basic',
    'username' => 'root',
    'password' => '',
    'charset' => 'utf8',
];
```

For an application that is aware of its environment, however, we should replace this file with a configuration that will use the `APPLICATION_ENV` variable that we defined earlier:

```
<?php return require __DIR__ . '/env/' . APPLICATION_ENV .
'/db.php';
```

Right now, we're just setting things up. We'll cover how to set up our directories in the next section.

With this change, our application now knows that it needs to look in a file called `db.php` under `config/env/<APPLICATION_ENV>/` to pull the correct configuration environment for that file.

Parameter configuration

In a manner similar to our database configuration file, Yii also lets us use a parameter file where we can store all of the noncomponent parameters for our application. This file is located at `config/params.php`. Since the basic app doesn't make this file aware of its environment, we'll change it to do that as follows:

```
<?php return require __DIR__ . '/env/' . APPLICATION_ENV .
'/params.php';
```

Environment configuration

Finally, we have the environment configuration that we defined earlier when working with our entry scripts. We'll store this file in config/env.php, and it should be written as follows:

```
<?php return require __DIR__ . '/env/' . APPLICATION_ENV .
'/env.php';
```

Most modern applications have several different environments depending upon their requirements. Typically, we'd break them down into four distinct environments:

- The first environment we typically have is called **DEV**. This environment is where all of our local development occurs. Typically, developers have complete control over this environment and can change it, as required, to build their applications.
- The second environment that we typically have is a testing environment called **TEST**. Normally, we'd deploy our application to this environment in order to make sure that our code works in a production-like setting; however, we normally would still have high log levels and debug information available to us when using this environment.
- The third environment we typically have is called **UAT**, or the User Acceptance Testing environment. This is a separate environment that we'd provide to our client or business stakeholders for them to test the application to verify that it does what they want it to do.
- Finally, in our typical setup, we'd have our **PROD** or production environment. This is where our code finally gets deployed to and where all of our users ultimately interact with our application.

As outlined in the previous sections, we've been pointing all of our environment configuration files to the `config/env/<env>` folder. Since our local environment is going to be called `DEV`, we'll create it first:

1. We'll start by creating our `DEV` environment folder from the command line:

   ```
   mkdir -p config/env/dev
   ```

2. Next, we'll create our `dev` database configuration file in `db.php` under `config/env/dev/`. For now, we'll stick with a basic SQLite database:

   ```
   <?php return [
       'dsn' => 'sqlite:/' . __DIR__ .
       '/../../../runtime/db.sqlite',
         'class' => 'yii\db\Connection',
       'charset' => 'utf8'
   ];
   ```

3. Next, we'll create our environment configuration file in `env.php` under `config/env/dev`. If you recall from earlier in the chapter, this is where our `debug` flag was stored, so this file will look as follows:

   ```
   <?php return [
       'debug' => true
   ];
   ```

4. Finally, we'll create our `params.php` file under `config/env/dev/`. As of now, this file will simply return an empty array:

   ```
   <?php return [];
   ```

Now, for simplicity, let's copy over this configuration to our other environments. From the command line, we can do that as follows:

```
cp -R config/env/dev config/env/test
cp -R config/env/dev config/env/uat
cp -R config/env/dev config/env/prod
```

Setting up our application environment

Now that we've told Yii what files and configurations it needs to use for each environment, we need to tell it what environment to use. To do this, we'll set custom variables in our web server configuration that will pass this option to Yii.

Setting the web environment for NGINX

With our console application properly configured, we now need to configure our web server to pass the `APPLICATION_ENV` variable to our application. In a typical NGINX configuration, we have a location block that looks as follows:

```
location ~ \.php$ {
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME $document_root/
        $fastcgi_script_name;
        fastcgi_pass   127.0.0.1:9000;
        #fastcgi_pass unix:/var/run/php5-fpm.sock;
        try_files $uri =404;
    }
```

To pass the `APPLICATION_ENV` variable to our application, all we need to do is define a new `fastcgi_param` as follows:

```
fastcgi_param   APPLICATION_ENV "dev";
```

After making this change, simply restart NGINX.

Setting the web environment for Apache

We can also easily configure Apache to pass the `APPLICATION_ENV` variable to our application. With Apache, we typically have a VirtualHost block that looks as follows:

```
# Set document root to be "basic/web"
DocumentRoot "path/to/basic/web"

<Directory "path/to/basic/web">
    # use mod_rewrite for pretty URL support
    RewriteEngine on
    # If a directory or a file exists, use the request directly
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    # Otherwise forward the request to index.php
    RewriteRule . index.php

    # ...other settings...
</Directory>
```

To pass the `APPLICATION_ENV` variable to our application, all we need to do is use the `SetEnv` command as follows, which can be placed anywhere in our VirtualHost block:

```
SetEnv   APPLICATION_ENV dev
```

After making this change, simply restart Apache and navigate to your application.

At the most basic level, our application isn't doing anything different from what it was when we first ran the composer `create-project` command. Despite not doing anything different, our application is now significantly more powerful and flexible than it was before our changes. Later on in the book, we'll take a look at how these changes in particular can make automated deployments of our application a seamless and simple process.

Components and objects

There are two base classes that almost everything in Yii2 extends from: the `Component` class and the `Object` class.

Components

In Yii2, the `Component` class has replaced the `CComponent` class from Yii1. In Yii1, components act as service locators that host a specific set of application components that provide different services for the processing of requests. Each component in Yii2 can be accessed using the following syntax:

```
Yii::$app->componentID
```

For example, the database component can be accessed using this:

```
Yii::$app->db
```

The cache component can be accessed using this:

```
Yii::$app->cache
```

Yii2 automatically registers each component at runtime via the application configuration that we mentioned in the previous section by name.

To improve performance in Yii2 applications, components are lazy-loaded or only instantiated the first time they are accessed. This means that if the cache component is never used in your application code, the cache component will never be loaded. At times, however, this can be nonideal, so to force load a component, you can bootstrap it by adding it to the bootstrap configuration option in either `config/web.php` or `config/console.php`. For instance, if we want to bootstrap the log component, we can do that as follows:

```
<?php return [
    'bootstrap' => [
        'log'
    ],
    [...]
]
```

The `bootstrap` option behaves in a manner similar to the preload option in Yii1—any component that you want or need to be instantiated on bootstrap will be loaded if it is in the `bootstrap` section of your configuration file.

For more information on service locators and components, ensure that you read the *Definitive Guide to Yii* guide located at `http://www.yiiframework.com/doc-2.0/guide-concept-service-locator.html` and `http://www.yiiframework.com/doc-2.0/guide-structure-application-components.html`.